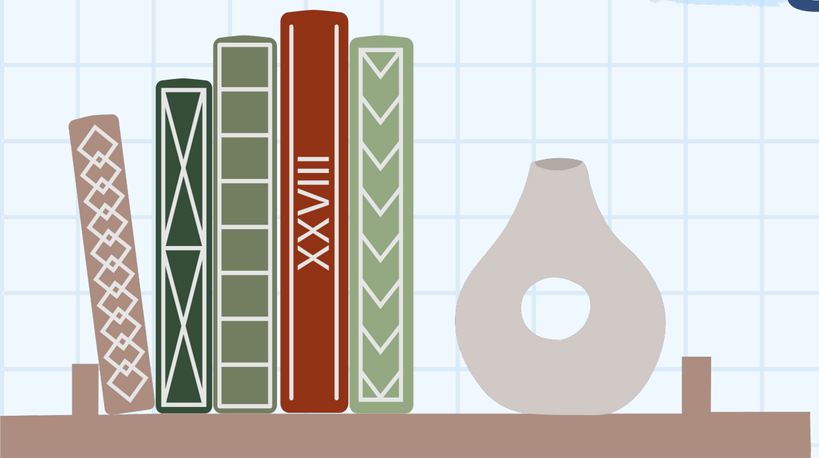
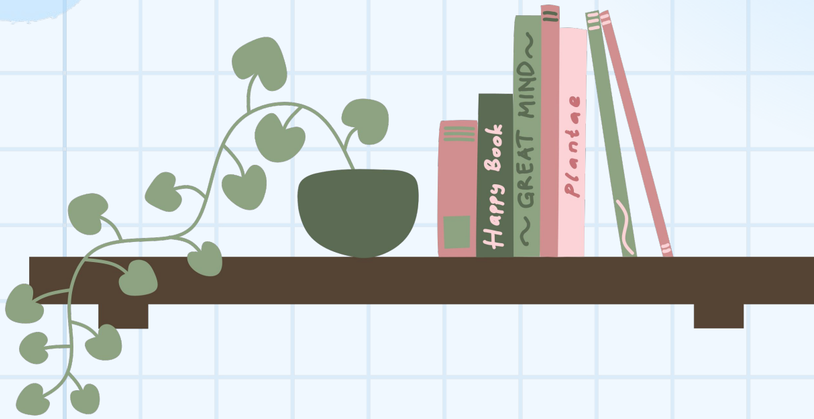
BS./BSC.

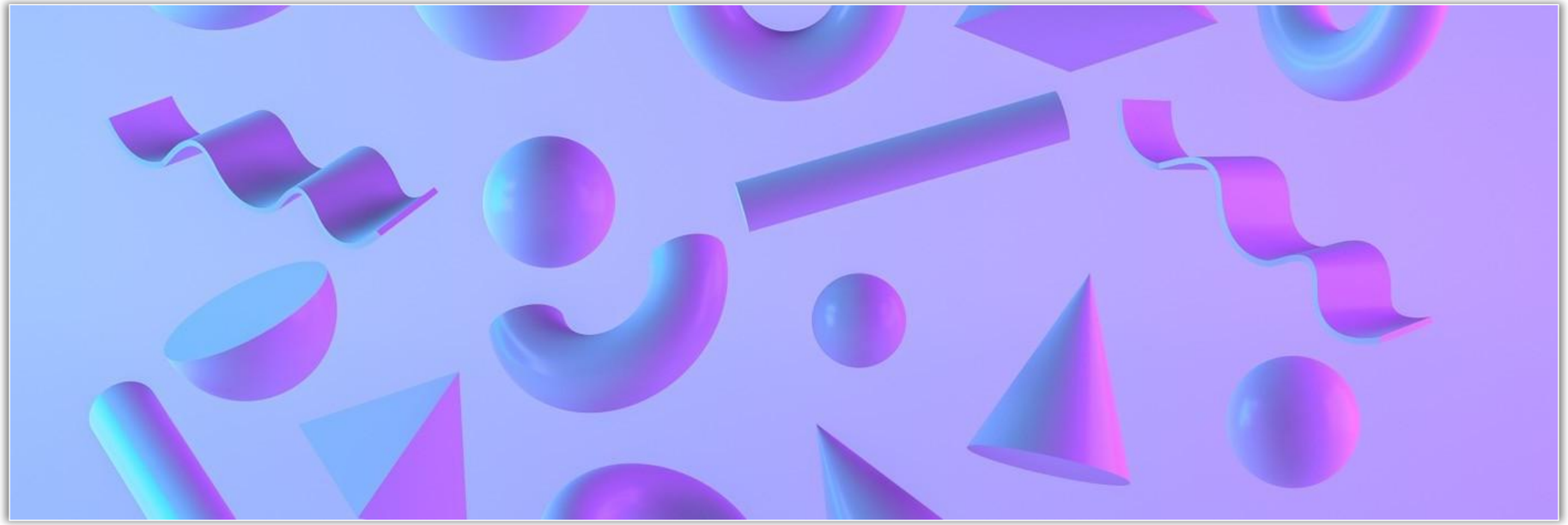
Applied AI and Data Science

Algorithmic Thinking & its Applications





Powered by Futurense



Types

Slide Credits : Cornell University

Review: **Types**

Type: set of values & operations on them

Type **float:**

- Values: real numbers
- Ops: +, -, *, /, //, %, **

Type **int:**

- Values: integers
- Ops: +, -, *, /, //, %, **

Type **str:**

- Values: strings
  - Double quotes: `"abc"`
  - Single quotes: `'abc'`
- Ops: + (concatenation)

Type **bool:**

- Values: `True`, `False`
- Ops: `not`, `and`, `or`

Types matter!

The programmer must decide: What is the right type for my data?

- Zip Code as an `int`?
- Grades as an `int`?
- Interest level as `bool` or `float`?

And, the programmer must decide whether to **convert** between types…

Type Conversions

Converting from one type to another

FYI some authors write "casting" instead of "conversion"

```
>>> float(2)
2.0


>>> int(2.6)
2




>>> type(2)
<class 'int'>
```

converts value `2` to type `float`

converts value `2.6` to type `int`

...different from:

`type(value)`

which *tells you* the

Can you guess what does Python does?

```
>>> 1/2.6
```

(A) turn 2.6 into the integer 2, then calculate 1/2 → 0.5

(B) turn 2.6 into the integer 2, then calculate 1//2 → 0

(C) turn 1 into the float 1.0, then calculate 1.0/2.6 → 0.3846…

(D) Produce a `TypeError` telling you it cannot do this.

(E) Crash your computer

Memorization is rarely what's important.

Knowing how to solve the problem is what's important.

Here that means how to **check** and **convert** types if needed.

Widening Conversions

From a **narrower** type to a **wider** type

`int` → `float`

Width refers to information capacity.
"Wide" means more information capacity

Python does automatically if needed:

- Example: `1/2.0` evaluates to a `float`:

```
0.5
```


Note: does not work for **`str`**

- Example: `2 + "ab"` produces a `TypeError`

Narrowing Conversions

From a **wider** type to a **narrower** type

`float  int`

- Causes information to be lost
- Python **never** does this automatically

```
>>> 1 + 2.6
3.6
>>> 1 + int(2.6)
3
```

Operator Precedence

(a reminder from grade school math)

Operator Precedence

What is the difference between:

2*(1+3)	2*1 + 3
add, then multiply	***multiply, then add***

Operations performed in a set order

- Parentheses make the order explicit

What if there are no parentheses?

 Operator Precedence: fixed order to process operators when no parentheses

Precedence of Python Operators

- **Parentheses: (…)**
- **Exponentiation**: **
- **Negation**: –
- **Arithmetic**: * / // %
- **Arithmetic**: +–
- **Comparisons**: < > <=>=
- **Equality relations**: == !=
- **Logical not**
- **Logical and**
- **Logical or**

- **Precedence** goes downwards
  - Parentheses highest
  - Logical or lowest
- Same line means same precedence
  - Read "ties" left to right (except for **)
  - Example: 1/2*3 is (1/2)*3

But how do we know that? We read the documentation!

https://docs.python.org/3.12/reference/expressions.html#operator-precedence

Memorization is rarely what's important.

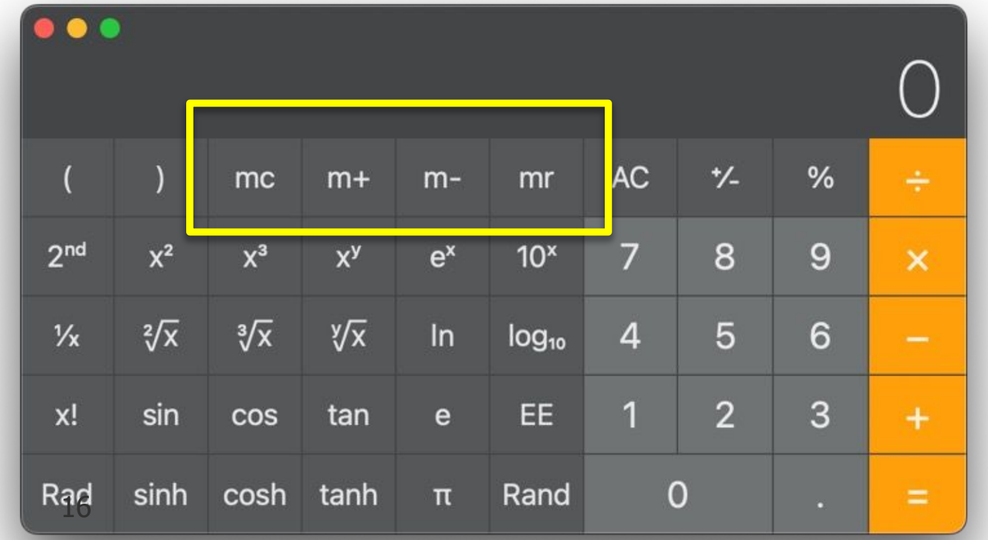
Knowing how to solve the problem is what's important.

Here that means how to **find and understand documentation** about operators.

Variables and Assignment

Many calculators have memory

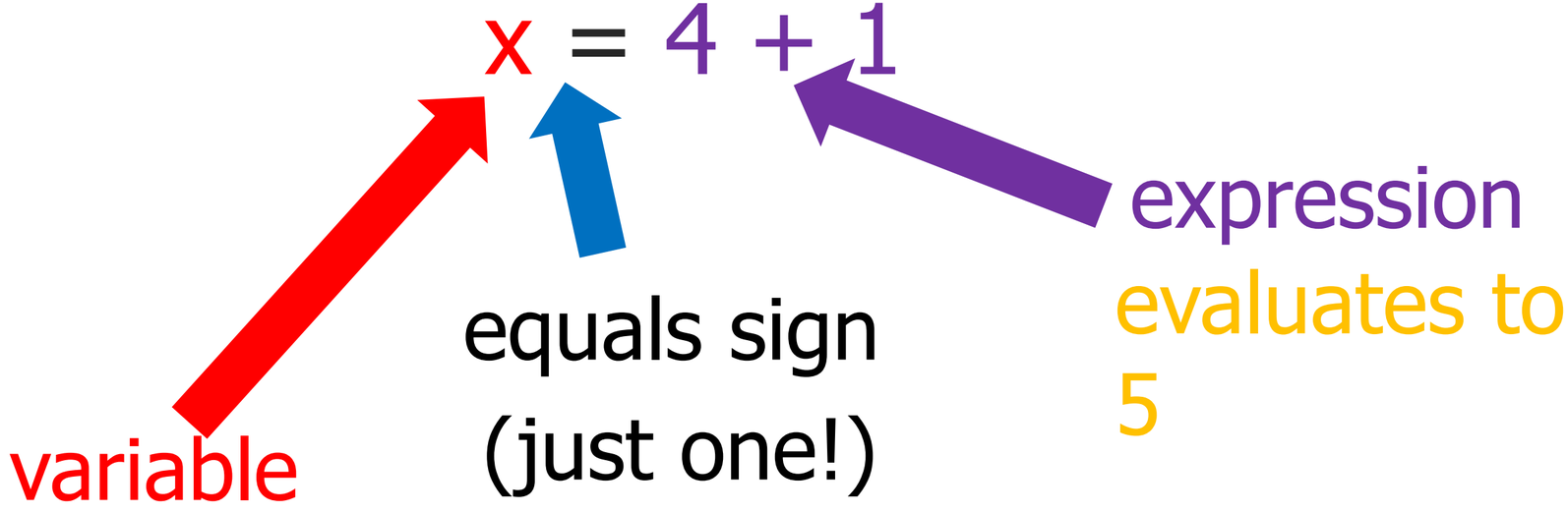
Casio calculator



Mac calculator

Computers have memory

Assignment statement:



An *assignment statement:*

- takes an *expression*
- evaluates it, and
- stores the *value* in a *variable*

Assignment statement

x = 4 + 1

How to pronounce that equals sign:

- x "gets" 4 + 1
- x "becomes" 4 + 1
- x "is assigned" 4 + 1
- x "is set to be" 4 + 1

But not:

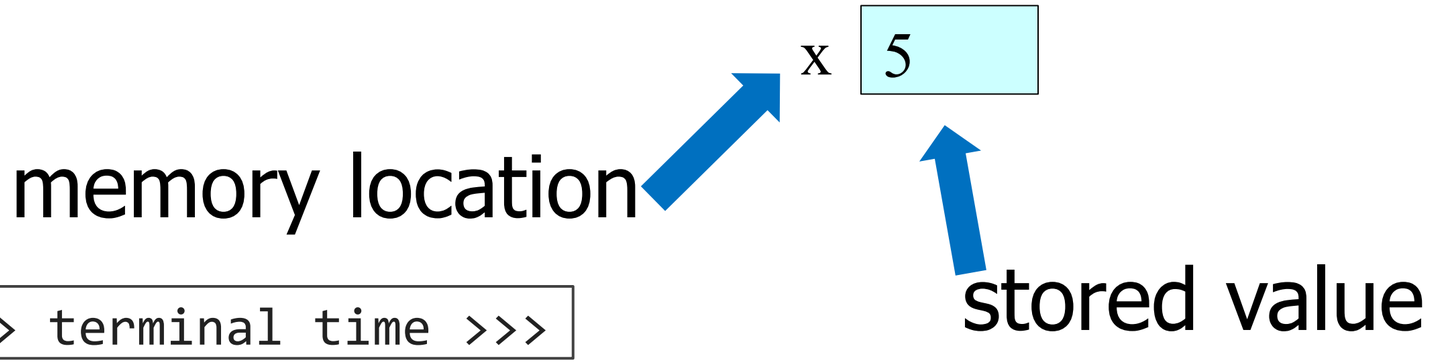
Assigning variables in interactive mode

```
>>> x = 5
>>>
```

Press ENTER and…

Hmm, looks like nothing happened…

- But something did happen!
- Python *assigned* the *value* 5 to the *variable* x
- Internally (and invisible to you):



Retrieving variables in interactive mode

```
>>> x = 5
>>> x
5
```

Press ENTER and…

Interactive mode displays the value of x

```
>>> terminal time >>>
```

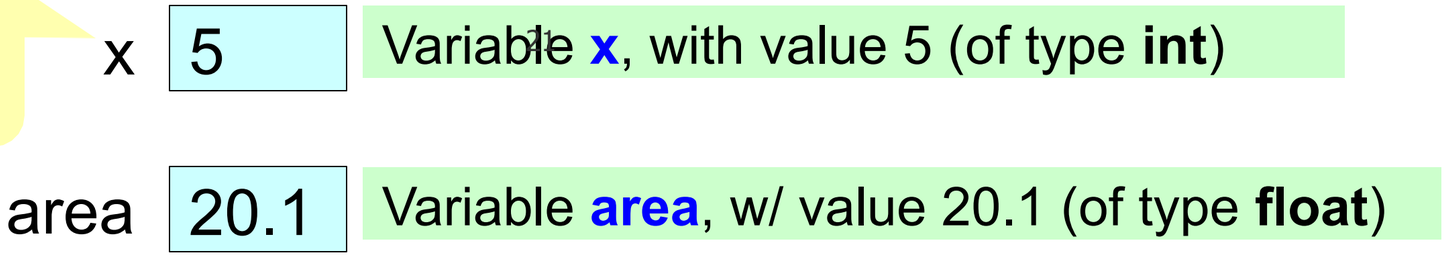
Variables

- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)

- Examples:

The type is a property of the *value*, not the *variable*.

Variable names usually must start with a letter.



Statements

```
>>> x = 5
>>>
```

Press ENTER and…

Hm, looks like nothing happened…

- This is a **statement**, not an **expression**
  - Tells the computer to do something (not give a value)
  - Typing it into >>> gets no response (but it is working)

Statements vs. Expressions

Expression

- **Does** something
  - Python *executes it*
  - End result is an action
  - Analogy: verbs
- Examples:
  - x = 2 + 1
  - x = 5

Statement

- **Represents** something
  - Python *evaluates it*
  - End result is a value
  - Analogy: nouns
- Examples:
  - 2.3
  - (3+5)/4
  - x == 5`

Look so similar ***but they are not!***

Visualizing Variables

The Python Tutor

https://pythontutor.com/

Try:

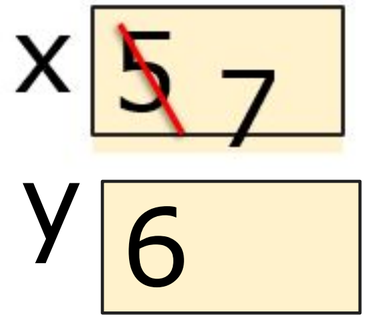
x =

5

y = 6

x = 7

Visualizing "on paper"



- Draw boxes on paper:

```
>>> x = 5
```

- New variable declared?

```
>>> y = 6
```

Write a new box.

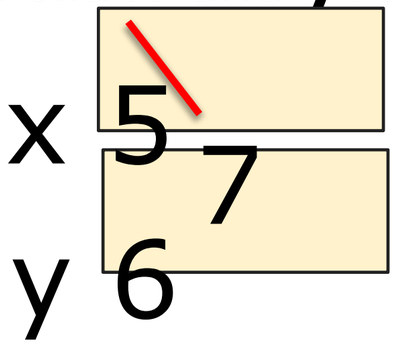
- Variable updated?

```
>>> x = 7
```

Cross out old value. Insert new value.

State diagrams

- We call these visualizations **state diagrams**
- It's a carefully designed graphical notation (and we haven't seen all of it yet) that reflects the reality of hardware
- We use this notation on slides and **you must use it on exams**



- We're not being silly! Being able to draw these diagrams means that:
  - You understand how to read Python code
  - You can explain Python code to yourself
  - You can demonstrate your understanding to others in this class
- The textbook and the Python Tutor use slightly different graphical notations to convey the same information; but on exams **you must use our slide notation** for state diagrams

Task: Draw state diagram for:

```
x = 5
x = x + 2
```

1. First statement: "x gets 5". Draw it on paper:

2. Second statement: "x gets x plus 2"
   - Evaluate the RHS expression, **x + 2**
   - Store the resulting value in the variable named on LHS, **x**
     - Cross off the old value in the box
     - Write the new value in the box

Which one is closest to your answer?

A.

x 5 7

B.

x 5

x 7

C.

x 5

x 7

D.

¯_(ツ)_/¯

x = 5

x = x + 2

Dynamic Typing

Dynamic Typing

The following is acceptable in Python:

```
>>> x = 1
>>> x = x / 2.0
```

 x contains an **int** value

 x now contains a **float** value

Python is a **dynamically typed** language in which:

- Any type of value can be stored in each variable
- Each variable can hold a value of a different type at different times

Alternative: a **statically typed** language

- Examples: Java, C
- Each variable restricted to values of just one type

Testing Types

Command: `type(value)`

Can query a variable's type:

```
>>> x = 5
>>> type(x)
<class 'int'>
```

Can compare the type of an expression:

```
>>> type(2) == int
True
```

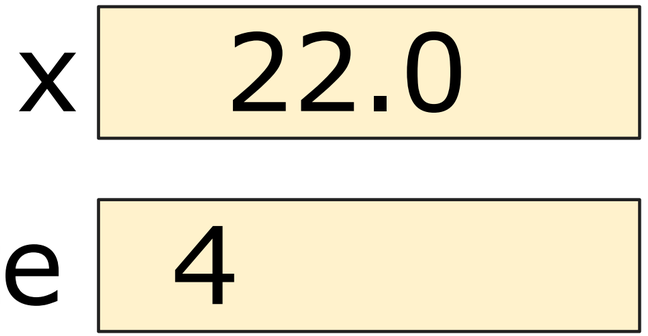
Additional Examples

Example 1

Have variable **x** already from previous

Create a new variable:

```
>>> rate = 4
```



Execute this assignment:

```
>>> rate = x / rate
```

Which one is closest to your answer?

A.
x 22.0 5.5
rate 4 5.5

B
x 22.0
rate 4
rate 5.5

C
x 22.0
rate 4 5.5

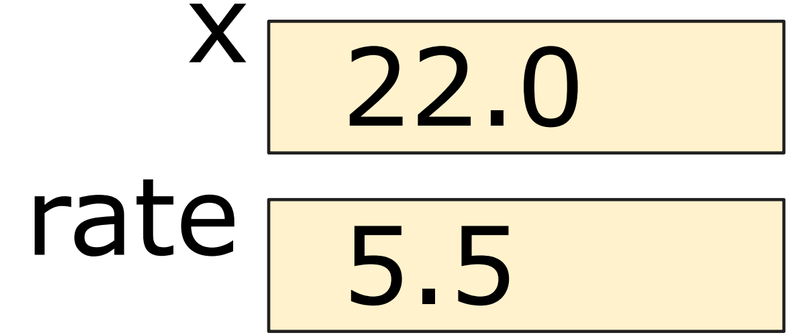
D
x 22.0
rate 4 5

E.
¯_(ツ)_/¯

```
rate = x / rate
```

Example 2

Begin with:



Execute this assignment:

```
>>> rat = x + rate
```

Which one is closest to your answer?

A.
x 22.027.5

rate 5.5

B
x 22.0

rate 5.5

rat 27.5

C
x 22.0

rate 5.5 27.5

D
x 22.0

rate 5.5

rat 27.5

E. ¯_(ツ)_/¯

```
rat = x + rate
```

Acknowledgments

- CS 1110 slide development by E. Andersen, A. Bracy, M. Clarkson, D. Gries, L. Lee, S. Marschner, W. White
- Vector art of cacti licensed from Vecteezy for use in CS 1110 Spring 2025
- Images not otherwise attributed are licensed from Microsoft 360 stock media library
- Photo of Casio calculator memory buttons: https://www.casio- intl.com/asia/en/calc/use/movie_1/

Thank you